INTRODUCTION TO COMPUTER SCIENCE I
LAB-1
A First Program (Or Two)

1 Getting started

The following steps will get you started with this lab. You will be introduced to the programs and tools that you will need to complete the lab and project assignments.

1. **Login to a server:** The computer systems that we will use for our projects are `romulus.amherst.edu` or `remus.amherst.edu`, (henceforth, `remus/romulus`), which are UNIX (Linux) systems.[1] To use these systems, you must login to them from your computer, using software known as an *X11 Windows Server*. Installing and starting this software is modestly different from one type of computer to another:

   - **Windows:** Follow these instructions to install and run *VcXsrv* for the first time.

     (a) **Install:** (**If you are working on one of the lab's workstations, skip this step.**) Go to the web site for *VcXsrv*. Click the large and obvious button labeled, *Download*. Your browser will download a file named something like:

     `vcxsrv-64.1.20.1.4.installer.exe`

     When this file is completely downloaded, run it. The installer will ask permission to install the software; click *Yes*, and accept any defaults about where to put the program, etc.

     (b) **Download configuration(s):** Go to my *VcXsrv* web page, where there are *XLaunch configuration files*—one for connecting to each server. Download one or both by **right-clicking** on it and then selecting *Save link as...* I recommend saving these to your `Desktop` folder for easy access.

     (c) **Launch:** On your desktop (or wherever you saved them), double-click on one of the XLaunch configuration files to launch it.

     (d) **Connect:** The first time you connect to each server (and only the first time), you will see a window asking you if you want to accept the server's credentials/keys. Type `y` and press enter.

[1] These servers are identical in all but name: no matter which one you are using, you will see the same files and use the same programs. There are two of them simply to spread the load of so many students connecting at one time.

A window will then prompt you for your **username**. Enter it. Note that (somewhat strangely) asterisks will appear, hiding what you're typing.

One more window will prompt you for your **password**. Enter it. Again (and this time, more sensibly), asterisks will appear in place of what you type.

After all of these steps, you will be presented with an *xterm terminal window*, in which you will be presented with a *shell prompt*. Continue onto the next step.

- **MacOS:** Follow these instructions to install and run *XQuartz* for the first time:

  (a) **Install:** Go to the website for XQuartz. There, download the installer shown under the header *Quick Download*, named something like:

  `XQuartz-2.7.11.dmg`

  Open this file, which will start the installer. Follow the instructions and accept the default options to install *XQuartz*.

  (b) **Launch:** When the installation is complete, search for `XQuartz` and run it.

  From the menu bar, click on the *File* menu, and then select *Terminal*. A *terminal window* will appear, and within it there will be a *shell prompt*, at which you can type commands.

  (c) **Connect:** Enter the following command to connect to one of the servers. Replace my username with **your username** before the `@` symbol; you may connect to `remus` as shown, or replace that with `romulus` to connect with that server:

  ```
  $ ssh -Y sfkaplan@remus.amherst.edu
  ```

  The first time you connect to each server, you will be asked whether you want to *accept a key*—a special, cryptographic value used to keep your communication with the server secure. Just enter `yes` and press enter.

  You will then be prompted for your **password**, so enter it. Nothing will appear as you type, hiding your password from anyone looking at your screen.

  In your terminal window, you will see a new shell prompt that shows that you are connected to the server that you chose. Move onto the next step.

2. **The *shell*:** In your *terminal window*, you will see a prompt presented by a *shell*—a program to which you type commands to the system. The shell interprets your commands and

launches the program your request. You will use it to start a *text editor* to see and alter your source code, to run a *compiler* that checks and transforms your source code into a program that can run, and to *execute (run)* the programs that you create.

As a first command, just to try things out, try the `whoami` command. That is, at your prompt, type that command and press enter; the program's response (which should be your username) should appear. It would look something like this:

```
[sfkaplan@remus ~/Desktop]$ whoami
sfkaplan
```

Notice that everything up to and including the dollar-sign (`$`) is the *shell prompt*—what the shell prints to the terminal to show you that it's ready for a command. You type only what follows. Henceforth, in these lab/project documents, I will use just the dollar-sign to indicate the prompt, even though your shell likely prints some stuff (e.g., your username, the server to which you're connected) prior to that dollar-sign.

As a second command, see who else is connected to the server with the `who` command, which would look something like this:

```
$ who
root      pts/0  2018-10-04 10:54 (sngtools03.amherst.edu)
sfkaplan  pts/1  2019-01-25 09:42 (temp6.cs-res.amherst.edu)
lamcgeoch pts/5  2018-11-26 19:00 (:279.0)
mriondato pts/22 2018-11-08 21:03 (:225.0)
```

3. **Make a directory:** When you first login and open a terminal window, you will be working in your *home directory*.[2] Your shell prompt (likely) shows you this with the part that shows a *tilde* (`~`). This symbol is used by the shell as a shorthand for your *home directory*—the directory in which all of your personal files and subdirectories/folders are stored. The prompt is thus showing you that you are currently working in your home directory.

   To create a directory for your work for this lab, use the `mkdir` (**m**a**k**e **dir**ectory) command, like this:

   ```
   $ mkdir lab-1
   ```

   Notice that this command produces no output: *no news is good news* with many commands.

   Next, use the `cd` (**c**hange **d**irectory) command to make that new folder the shell's current one, like this:

[2] *Directories* and *folders* are the same thing.

```
$ cd lab-1
```

You should notice that your shell prompt changes, showing the current directory as something like: `~/lab-1`

4. **Get some starting source code:** Use the following command to obtain a Java source code file:

```
$ wget -nv -i https://bit.ly/COSC-111-lab-1-source
```

This command will generate some output to show you what it's doing. To ensure that you have copied the file into your `lab-1` subdirectory, use the `ls` (**ls**t directory) command to list the files in the current directory, noting that the character following the dash (`-`) is a lowercase letter `L`, and **not** the numeral `1`. The `-l` part of the command indicates that you want to list the directory using the *long format*:

```
$ ls -l
```

You should see an output that looks something like this:

```
total 6
-rw-r----- 1 sfkaplan sfkaplan 813 Jan 30 13:06 Calculations.java
-rw-r----- 1 sfkaplan sfkaplan 198 Jan 30 13:04 lab-1-file-list.txt
-rw-r----- 1 sfkaplan sfkaplan 359 Jan 13 16:47 Messages.java
```

There are three files here, one listed on each line. We can ignore some of the information shown, but we can grasp some of it now. Specifically, we see who owns the file (your username will be there), how long the file is (e.g., `813` bytes), the date and time of the most recent update to the file, and the file's name.

You have now set yourself up to work with some Java source code. Move onto the next section.

2 Your assignment

2.1 Messages

Perform the following steps in order to see, use, and change the *Messages* program:

1. **Examine the source code:** Run *Emacs*, a programming text editor, to examine the `Messages.java` file. In the following command, be sure to include the trailing *ampersand* (`&`), causing the text editor to run in the *background*—that is, to run while allowing you to enter more commands:

```
$ emacs Messages.java &
```

A new window will open, showing you a handful of lines of Java code (along with *code comments*—lines that begin with two forward slashes (`//`)—that provide some human documentation and commentary. This code shows a program named `Messages` that will start on the line beginning `public static void main (String[] args)`, carry out two commands to print messages, and then end. For now, examine the code, but leave it as-is.

ASIDE: *Emacs* is a complex program that can do a great deal. To learn more about how to use it, you should use the built-in *Emacs tutorial*, which you can start within *Emacs* by first holding the `Ctrl` key and pressing `H`, and then releasing the `Ctrl` key and pressing `T`. **Seriously, you should do it.** It is a step-by-step introduction to so many helpful commands and keyboard shortcuts, and will make your work so much easier in this course; it is worth the up-front effort.[3]

2. **Compile the code:** The source code must be translated it into a form that the computer can execute. Leaving your *Emacs* window open, switch over to your terminal window again. In it, use the following command to *compile*—that is, translate into machine code—your source code:

   ```
   $ javac Messages.java
   ```

   Again, **no news is good news**. That is, if the computer simply presents the shell prompt to you after you issue this command, then **the compilation succeeded**. The *compiler*—the `javac` program—will print messages into your terminal window only if it was unable to translate your program. Given that you've changed nothing from the original code, it should compile without error.

3. **Execute your program:** Once you have successfully compiled your program, it is time to run it and see what happens.

   ```
   $ java Messages
   ```

   The messages that you saw in the code should be printed into your terminal window.

4. **Change something:** Alter your program slightly, and then test that your alteration did what you expected. Specifically:

   (a) **Edit:** Go back to your *Emacs* window. **Add your own print statement or two**, using the existing print statements as templates. Be sure to **save** your changed source code.

   (b) **Compile:** Issue the `javac` command to your shell, just like you did above. If there are error messages, you will need to return to *Emacs*, fix whatever is incorrect, save the changed code, and try to compile with `javac` again. Repeat until your code compiles with no error messages.

[3] Some of you may be familiar with *Integrated Development Environments (IDE's)* such as *DrJava*, *BlueJ*, *IntelliJ*, and *Eclipse*. You are welcome to try them if you like, but you will be on your own in using them. I recommend that you leave them alone and just use the tools presented here.

(c) **Execute:** Issue the `java` command to your shell, just like you did above. Your program should run, this time printing the additional message that you added into the code.

Ta-da! You've just done some programming. You have also begun to use the tools on which your work will depend in this course: your X11 windows server, remus/romulus, terminal windows and the shell, the *Emacs* text editor, the `javac` compiler, and the `java` executor.

2.2 Calculations

Now, turn your attention to the *Calculations* program. Start by using *Emacs* to see and edit the code:

```
$ emacs Calculations.java &
```

You will see that, in this program, the code comments note two places where you must add lines of code in order to perform some arithmetic calculations. Try to write these lines. Each time you add or otherwise change the code, you can test it as follows:

1. **Save:** Be sure to save your changed source code in *Emacs*.
2. **Compile:** Use the `javac` command to translate your source code into something the machine can understand:

   ```
   $ javac Calculations.java
   ```

3. **Run:** If no errors appear during compilation, make your program go:

   ```
   $ java Calculations
   ```

This sequence—*edit, save, compile, run*—is what you must repeat to complete, correct, and refine your code. Do so until the calculations are being done correctly, the program compiles without errors, and run with the correct results appearing.

3 How to submit your work

Submit your modified `Messages.java` and `Calculations.java` files using the `cssubmit` command. This command will prompt you for the course and assignment for which you are submitting work. It looks like this:

```
$ cssubmit Messages.java Calculations.java

Choose your course:

1)   COSC-111-02 (Prof. Kaplan)
2)   COSC-111-03 (Prof. McGeoch)
3)   Theoretical Foundations

Enter its number (1 to 3), or q to quit: 1

Choose your assignment:

1)  Lab-1: Getting started

Enter its number (1 to 1), or q to quit: 1
Do you really want to submit 1 file(s) to Lab-1: Getting started?
Enter y or n: y
2 file(s) successfully submitted
```

You may also access the CS submissions systems through your web browser at the submission system web page. Here, you can check whether and when you have submitted work for each assignment. Note that **you may submit work multiple times**; each submission is separately logged, and I will grade the most recent submission for each assignment.

This assignment is due on Thursday, Feb-07, 11:59 pm.